STRING HANDLING

1. **PROBLEM STATEMENT**

Character Transformation Task

Imagine you're developing a secure messaging application where users can send encoded messages. Your task is to implement a method to transform a given string into an encoded format based on specified rules. The encoded message is generated by subtracting the length of the string from the ASCII value of each character and converting these values back to characters. Display the transformed message.

Scenario Description:

You receive input that includes a message string in the `UserInterface` class.

Component Specifications: MessageEncoder (Utility class)

Type(Class)	Method	Parameters	Responsibilities
MessageEncoder	encodeMessage	String message	Encodes the given message by subtracting the length of the string from the ASCII value of each character.

Validation Rules:

1. String Length Check:
   - The length of the string must be greater than 4. If not, print "The string `<String>` has minimum length" and terminate the program.

2. Space Check:
   - The input string must not contain any spaces. If it does, print "The string `<String>` should not contain space" and terminate the program.

Sample Input/Output:

Sample Input 1:
```
Enter message:
Hello
```
Sample Output 1:
```
**@EBBI**
```
Explanation:

ASCII value equivalents:
H=72, e=101, l=108, l=108, o=111

Length of the given string "Hello":
5

Subtracting with ASCII equivalents:
72-5=67, 101-5=96, 108-5=103, 108-5=103, 111-5=106

Character Equivalents:
67=@, 96=``, 103=g, 103=g, 106=j

Encoded message:
@EBBI

Sample Input 2:
```
Enter message:
ABCDE
```
Sample Output 2:
```
**=>?@AB**
```

Sample Input 3:
```
Enter message:
HelpMe
```
Sample Output 3:
```
**WebPage**
```
Sample Input 4:
```
Enter message:
Good Morning
```
Sample Output 4:
```
**The string Good Morning should not contain space**
```

Sample Input 5:
```
Enter message:
Ok
```
Sample Output 5:
```
**The string Ok has minimum length**
```

==========

2. PROBLEM STATEMENT

Read two lines of information store it in s1 and s2 respectively. Input only alphabets, numbers and space if enter other this any special chars or anything else should say “Invalid Input”. Output prints the longest common substring length (including space), and print the longest common substring in upper and lower case.

==========

3. **PROBLEM STATEMENT**

You are tasked with developing a program for analyzing a set of N DNA sequences provided as input. Each sequence consists of characters 'A', 'T', 'C', and 'G'. The program should determine the count of positions in each sequence where either all characters are purines (A or G) or all characters are pyrimidines (C or T). Purines and pyrimidines are nucleotide bases in DNA.

Constraints:

- The number of sequences (N) should be greater than zero. If N is zero or negative, print "The number of sequences must be greater than zero" and terminate the program.
- All sequences must be of the same length. If they are of different lengths, print "All sequences are not of the same length" and terminate the program.

Input:

- The first line contains an integer N, the number of DNA sequences.
- The next N lines each contain one DNA sequence composed of characters 'A', 'T', 'C', and 'G'.

Output:

- N lines where each line contains two integers: the count of positions where all characters are purines and the count of positions where all characters are pyrimidines in the corresponding DNA sequence.

Sample Input / Output:

Sample Input 1:

```
3
ATGCTA
CGATCG
GGAATT
```

Sample Output 1:

```
**3 2**
```

Explanation:
- In sequence 1 (ATGCTA): Positions 1 (A), 4 (C), and 6 (A) are purines ('A' or 'G').
- In sequence 2 (CGATCG): Positions 2 (G) and 6 (G) are purines ('G').
- In sequence 3 (GGAATT): Positions 1 (G), 2 (G), and 3 (A) are purines ('G' and 'A').

Sample Input 2:
```
2
AGCTA
CGTA
```

Sample Output 2:
```
**5 3**
```

Explanation:
- In sequence 1 (AGCTA): Positions 1 (A), 2 (G), 3 (C), 4 (T), and 5 (A) are purines ('A' and 'G').
- In sequence 2 (CGTA): Positions 2 (G), 3 (C), and 4 (T) are purines ('G', 'C', and 'T').

Sample Input 3:
```
-2
```

Sample Output 3:
```
**The number of sequences must be greater than zero**
```

```

Explanation:
- The number of sequences is less than or equal to zero, hence the output indicates the error message.
==========

## 4. PROBLEM STATEMENT

You are developing a program to analyze the structure of N financial transaction identifiers provided as input. Each identifier is composed of a fixed number of alphanumeric characters. The program should identify positions in each identifier where all characters are digits ('0'-'9') or all characters are letters (either uppercase or lowercase). Finally, the program displays the count of such positions for each identifier.

**Constraints:**
- The number of identifiers (N) should be greater than zero. If N is zero or negative, print "The number of identifiers must be greater than zero" and terminate the program.
- All identifiers must be of the same length. If they differ in length, print "All identifiers are not of the same length" and terminate the program.

**Input:**
- The first line contains an integer N, the number of identifiers.
- The next N lines each contain one identifier composed of alphanumeric characters.

**Output:**
- N lines where each line contains two integers: the count of positions where all characters are digits and the count of positions where all characters are letters (either all uppercase or all lowercase) in the corresponding identifier.

***Sample Input / Output:***

**Sample Input 1:**
```
4
A1B2C3D4
5G6H7I8J9
abcd1234
XYZW9876
```
```

Sample Output 1:
```
**0 4**
```

Explanation:
- Identifier 1 (A1B2C3D4): No positions have all digits; Positions 1-4 have all letters ('ABCD').
- Identifier 2 (5G6H7I8J9): No positions have all digits; Positions 2-5 have all letters ('GHIJ').
- Identifier 3 (abcd1234): Positions 5-8 have all digits ('1234'); No positions have all letters.
- Identifier 4 (XYZW9876): Positions 5-8 have all digits ('9876'); Positions 1-4 have all letters ('XYZW').

Sample Input 2:
```
3
ABC123
DEF4567
GHIJKL
```

Sample Output 2:
```
**0 0**
```

Explanation:
- All identifiers are not of the same length (lengths are 6, 7, and 6 respectively), so the output is "All identifiers are not of the same length".

Sample Input 3:
```
-2
```

Sample Output 3:
```
**The number of identifiers must be greater than zero**
```

Explanation:

- The number of identifiers is less than or equal to zero, hence the output indicates the error message.

==========

5. PROBLEM STATEMENT

A local library wants to implement a new system for book checkouts. To enhance security, they decide to require patrons to enter a passphrase that meets a specific criteria based on the books they want to borrow. The passphrase must consist of words where the sum of the alphabetical positions of the characters in the first half of each word equals the sum of the positions in the second half.

Criteria:

- For each word in the passphrase, split it into two halves. If the word has an odd length, ignore the middle character.
- Calculate the sum of alphabetical positions (A=1, B=2, ..., Z=26) for each half.
- If the sums of both halves are equal, the word is considered valid.
- Otherwise, the word is considered invalid for the passphrase.

Write a program to implement this passphrase validation system. Your program should:

- Prompt the user to enter a passphrase containing multiple words separated by spaces.
- Validate each word according to the criteria.
- Output whether each word in the passphrase is valid or invalid.

Constraints:

- Each word in the passphrase must consist only of uppercase or lowercase English letters.
- Ignore any words that contain digits, special characters, or whitespaces.

Example:

Input:

```
Enter the passphrase: hello noon zips
```

Output:

```
hello is not a valid word
noon is a valid word
zips is not a valid word
```

Explanation:
- "hello": Split into "hel" and "lo". Left sum = 8 (h=8, e=5), Right sum = 15 (l=12, o=15). Not equal, so not valid.
- "noon": Split into "no" and "on". Left sum = 29 (n=14, o=15), Right sum = 29 (o=15, n=14). Equal, so valid.
- "zips": Contains characters other than letters, so it is not valid.

Your program should handle multiple words in the passphrase and provide clear output indicating which words meet the criteria.

==========

6. PROBLEM STATEMENT

The library management system is launching a new online portal for book reservations. To ensure security and accuracy, users need to authenticate using a PIN generated based on their membership ID. Develop a Java application to automate the PIN generation process.

Note:

1. Start by verifying the length of the membership ID. It should be exactly 8 characters long; otherwise, print "<length of membership ID> is an invalid length" and terminate the application.

2. The membership ID should consist of 3 alphabets followed by 5 digits. If this format is not followed, print "<membership ID> is an invalid format" and terminate the program.

3. The first digit of the numeric part of the membership ID (following the alphabets) should not be '0'. If it is '0', print "<first digit> is an invalid digit" and terminate the program.

4. The last digit of the numeric part of the membership ID should correspond to the age category of the member: 1 for ages 10-20, 2 for ages 21-30, and so on up to 9 for ages 80-90. If it doesn't match any category, print "<age category> is an invalid category" and terminate the program.

Logic of PIN Generation:

- The PIN will be the concatenation of the even-positioned digits (0-indexed) in the numeric part of the membership ID.

Example: Membership ID - ABC12345, Age Category - 2

Generated PIN: 24

Sample Input / Output:

Sample Input 1:

Enter the membership ID

ABC12345

Sample Output 1:

PIN: 24

Sample Input 2:

Enter the membership ID

ABCD123

Sample Output 2:

7 is an invalid length

Sample Input 3:

Enter the membership ID

ABC01234

Sample Output 3:

0 is an invalid digit

Sample Input 4:

Enter the membership ID

ABZ12345

Sample Output 4:

ABZ12345 is an invalid format

Sample Input 5:

Enter the membership ID

ABC12356

Sample Output 5:

6 is an invalid category

==========

7. **PROBLEM STATEMENT**

Students need to process a sentence where they are required to reverse the characters of each word that starts and ends with a vowel, while keeping other words unchanged. Help them implement this functionality using a Java program.

Requirements:

- If the sentence contains fewer than 2 words, print "Invalid sentence length".
- Words that start and end with vowels should have their characters reversed.
- Words should contain only alphabets and spaces; if not, print "<sentence> is an invalid sentence".

Sample Input/Output:

Sample Input/Output 1:

```
Enter the sentence
You are the apple of my eye
```
You are the elppa of my eye
```

Sample Input/Output 2:

```
Enter the sentence
Hello! How are you?
```
**Hello! How are you? is an invalid sentence**
```

Sample Input/Output 3:

```
Enter the sentence
I love OpenAI's models
Invalid sentence length
```